université PARIS DIDEROT PARIS 7

L2 informatique - Année 2018–2019

TD d'Éléments d'Algorithmique n° 11

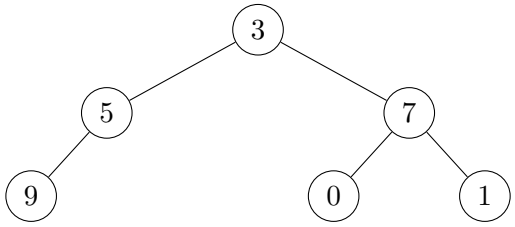
Exercice 1. *Parcours en profondeur itératif.*

1. Rappelez l'algorithme récursif pour le parcours en profondeur d'un arbre binaire, préfixe, infixe et postfixe.

Considérons maintenant l'algorithme itératif générique pour le parcours en profondeur d'un arbre binaire :

```
1
2  parcours_iteratif(arb:arbre){
3    if (arb.racine!=null) {
4      p = new Pile()
5      p.push((arb.racine,0))
6      while(p non vide) {
7        (a,x)=p.pop()
8        if(x=0) {
9          visiter_prefixe(a)
10         p.push((a,1))
11         if(a.G!=null) { p.push((a.G,0)) }
12       }
13       if(x=1) {
14         visiter_infixe(a)
15         p.push((a,2))
16         if(a.D!=null) { p.push((a.D,0)) }
17       }
18       if(x=2) {
19         visiter_suffixe(a)
20       }
21     }
22   }
23 }
```

et l'arbre binaire :



2. Exécutez l'algorithme sur l'arbre ci-dessus, dans le cas où la fonction `visiter_prefixe` affiche le champs `val` de son argument, et les fonctions `visiter_infixe` et `visiter_suffixe` ne font rien.
3. Même question, dans le cas où uniquement `visiter_infixe` affiche.
4. Même question, dans le cas où uniquement `visiter_suffixe` affiche.
5. Quel rapport entre l'algorithme récursif et itératif ?

Exercice 2. *Lecture d'algorithme.*

```
boolean rec (n: int, t: Arbre) {
  return rec_aux (n, t.racine)
}

boolean rec_aux(n: int, r: Noeud) {
  if r = null return false
  return  (r.val=n || rec_aux(n, r.G) || rec_aux(n, r.D))
}
```

1. Que calcule la fonction `rec` ci-dessus ?
2. Dans quel ordre explore-t-elle les nœuds de l'arbre `t` ? (l'expression `a||b` est évaluée comme `if a then true else b`).
3. Prouvez que la fonction `rec` effectue son calcul correctement.
4. Combien de comparaisons sont effectuées au maximum lors d'un appel de `rec` ?
5. Écrire une version itérative de `rec`, explorant les nœuds de `t` dans le même ordre.
6. Aurait-on pu utiliser l'algorithme de l'exercice 1 pour la question précédente ? Si oui, comment ?

Exercice 3. *Recherche d'un élément à profondeur minimale.*

Étant donnée une propriété P sur les entiers, on se propose d'écrire un algorithme qui prend un arbre binaire et renvoie un nœud de l'arbre de profondeur minimale parmi ceux dont la valeur vérifie P.

1. Pourquoi « un nœud », et pas « le nœud » ?
2. Quel type de parcours d'arbre est approprié pour resoudre ce problème ?
3. Écrire un algorithme pour la propriété $P(n)$: « n est un nombre premier ».

Exercice 4. *Arbres de Recherche.*

On dit qu'un arbre binaire est un *arbre binaire de recherche* s'il vérifie la propriété suivante : l'étiquette de tout nœud interne est supérieure ou égale a toutes les étiquettes des nœuds de son sous-arbre gauche, et inférieure à toutes les étiquettes des nœuds de son sous-arbre droit.

1. Écrire une fonction `rechABR(int v, Arbre a)` qui teste si la valeur `v` est présente dans l'arbre binaire de recherche `a`. Combien de comparaisons sont effectuées au maximum lors d'une recherche ?
2. Écrire une fonction `insere(Arbre a, int val)` qui ajoute un nouveau nœud de valeur `v` à l'arbre `a` de telle façon que si `a` est un ABR avant l'appel de fonction, alors `a` est encore un ABR après.
3. Partir d'un arbre vide et insérer les valeurs 4,2,1,3,5 dans cet ordre. Quel est l'arbre obtenu ?
4. Proposer un algorithme de tri d'une liste de valeurs en utilisant la fonction précédente et un parcours d'arbre.
5. Écrire une fonction `check(Arbre a)` qui permet de vérifier si un arbre `a` est un ABR. (Indication : on pourra faire une fonction récursive qui retourne le min et max de l'arbre si c'est bien un ABR.)